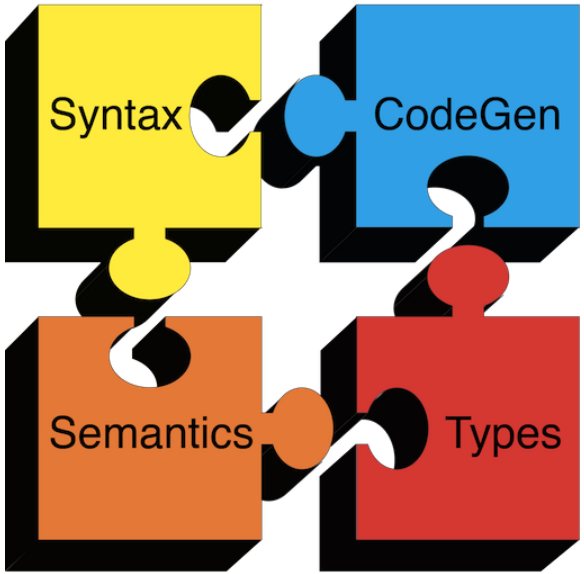


Compilers

Bottom-Up Parsing

Bottom-Up Parsing

- Bottom-up parsing is more general than (deterministic) top-down parsing
  - And just as efficient
  - Builds on ideas in top-down parsing

- Bottom-up is the preferred method

Bottom-Up Parsing

- Bottom-up parsers don't need left-factored grammars

- Revert to the "natural" grammar for our example:

  $E \rightarrow T + E \mid T$

  $T \rightarrow \text{int} * T \mid \text{int} \mid (E)$

- Consider the string: int * int + int

Bottom-up parsing *reduces* a string to the start symbol by inverting productions

int * int + int	$T \rightarrow int$
int * T + int	$T \rightarrow int * T$
T + int	$T \rightarrow int$
T + T	$E \rightarrow T$
T + E	$E \rightarrow T + E$
E	

Note the productions, read backwards, trace a rightmost derivation

int * int + int	$T \rightarrow$ int
int * T + int	$T \rightarrow$ int * T
T + int	$T \rightarrow$ int
T + T	$E \rightarrow T$
T + E	$E \rightarrow T + E$
E	

Bottom-Up Parsing

Important Fact #1 about bottom-up parsing:

A bottom-up parser traces a rightmost derivation in reverse

Bottom-Up Parsing

int * int + int

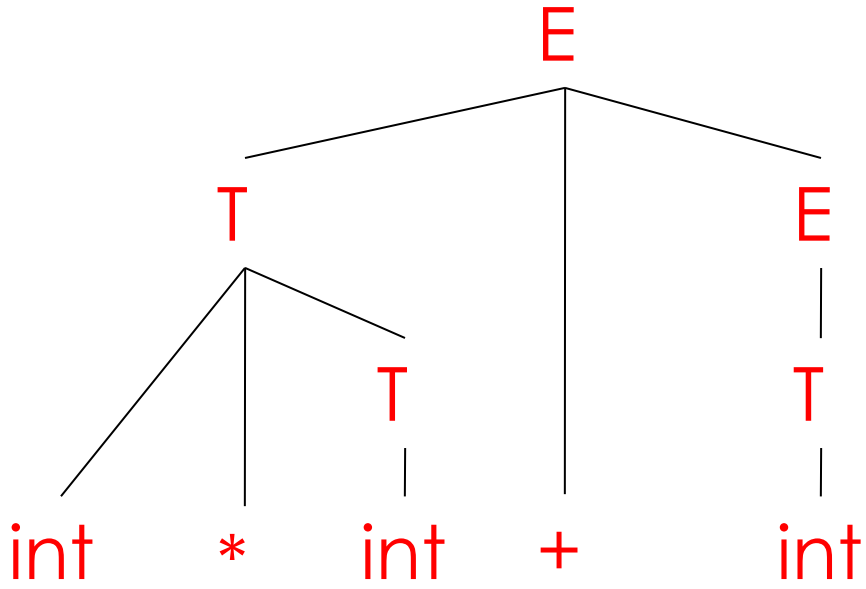
int * T + int

T + int

T + T

T + E

E



Bottom-Up Parsing

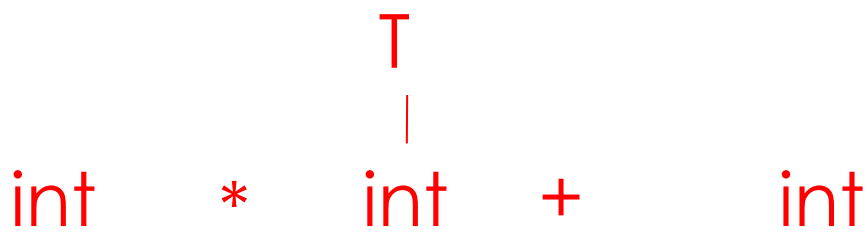
int * int + int

int * int + int

Bottom-Up Parsing

int * int + int

int * T + int

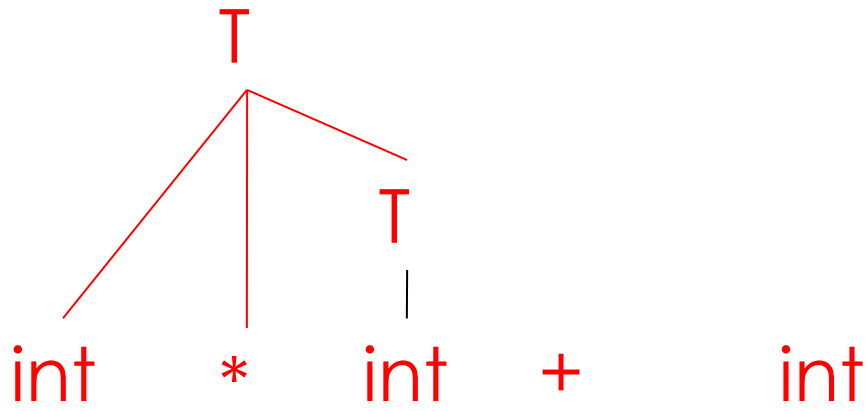


Bottom-Up Parsing

int * int + int

int * T + int

T + int



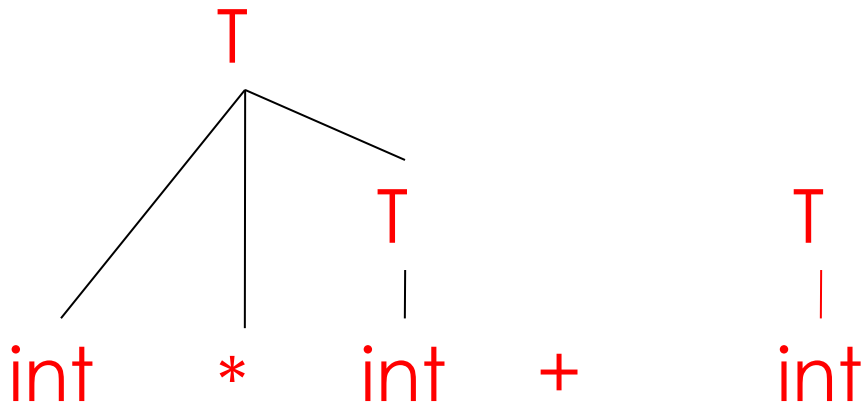
Bottom-Up Parsing

int * int + int

int * T + int

T + int

T + T



Bottom-Up Parsing

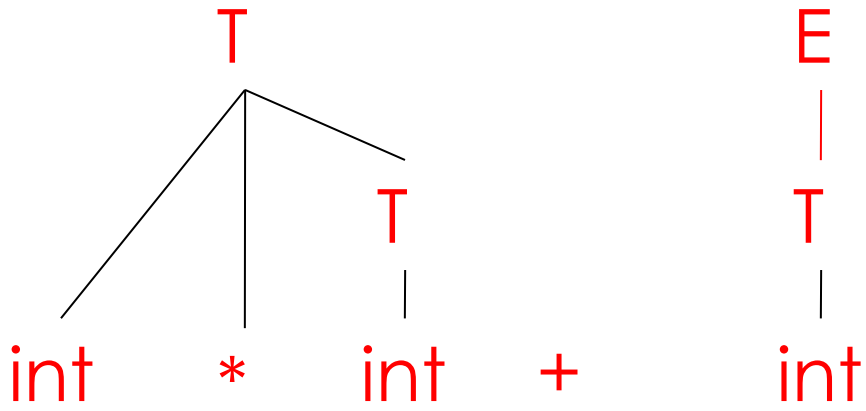
int * int + int

int * T + int

T + int

T + T

T + E



Bottom-Up Parsing

int * int + int

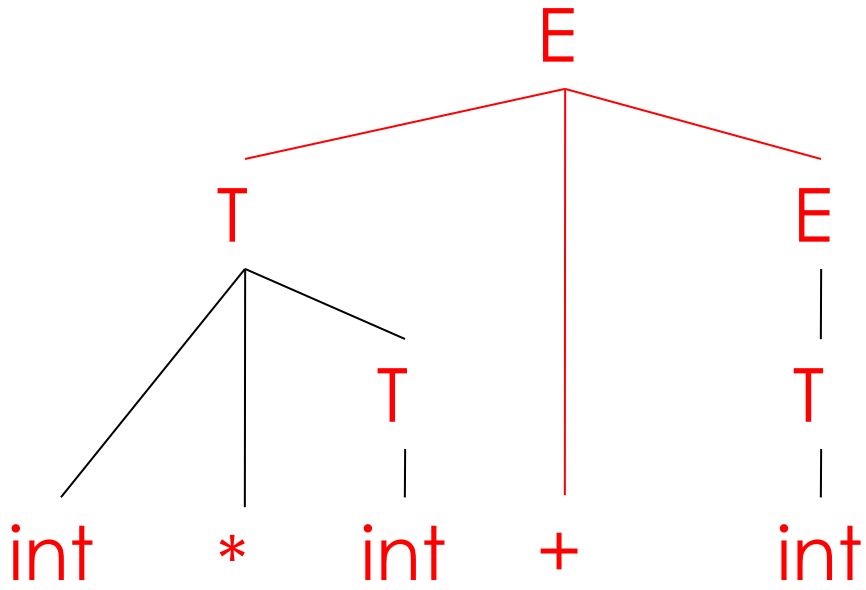
int * T + int

T + int

T + T

T + E

E



Bottom-Up Parsing

For the given grammar, what is the correct series of reductions for the string: -(id + id) + id

$E \rightarrow E' \mid E' + E$

$E' \rightarrow -E' \mid id \mid (E)$